Exemplar selection via leave-one-out kernel averaged gradient descent and Subtractive Clustering

Yiannis Kokkinos and Konstantinos G. Margaritis*

Parallel and Distributed Processing Laboratory, Department of Applied Informatics, University of Macedonia, 156 Egnatia str., P.O. Box 1591, 54006, Thessaloniki, Greece

Keywords. Subtractive Clustering, kernel averaged, leave-one-out, gradient descent, automatic exemplar selection

Abstract. Scalable data mining and machine learning require data abstractions. This work presents a scheme for automatic selection of representative real data points as exemplars. Currently few algorithms can select representative exemplars from the data. K-medoids and Affinity Propagation are such algorithms. K-medoids requires the number of exemplars to be given in advance, as well as a dissimilarity matrix in memory. Affinity propagation automatically finds exemplars as well as their *k* number but it requires a similarity matrix in memory. A fast algorithm, which works without the need of any matrix in memory, is Subtractive Clustering, but it requires user-defined bandwidth parameters. The essence of the proposed solution relies on a leave-one-out kernel averaged gradient descent that automatically estimates a suitable bandwidth parameter from the data in conjunction with Subtractive Clustering algorithm that further uses this bandwidth for extracting the most representative exemplars, without initial knowledge of their number. Experimental simulations and comparisons of the proposed solution with Affinity propagation exemplar selection on various benchmark datasets seem promising.

1 Introduction

A common problem in applications that collect and store their data is that the number of training examples may be large. Hence, many machine learning and data mining algorithms become slow [1][2]. One of the solutions is to select most representative exemplars from the data. These exemplars are real data points that form an abstract view of the whole dataset, can represent the structure of the data and can also be used for recognizing patterns [2]. Finding exemplars is a hard problem [3] but is more interesting and informative than dividing data into clusters. Detecting exemplars goes beyond simple clustering, as the exemplars store compressed information [3]. Hence, exemplar selection techniques try to find additional regional information in order to extract representative *k*-exemplars or *k*-medoids or *k*-centers which are close to any given training point so as to minimize the maximum distance from a point to its nearest exemplar. The first exemplar-based algorithm was *k*-medoids [4] which requires the number *k* of exemplars to be given in advance, as well as a dissimilarity matrix in

memory. Yet, finding exemplars without knowing the k number is a challenge since this k-centers problem or k median objective is NP hard [5] **[**6**]** [7].

Currently, Affinity Propagation (AP) introduced by Frey and Dueck [8] is the state-of-the-art algorithm for detecting exemplars and subsequently clustering the data around them. AP has been applied in various fields and many applications. In AP all data points are simultaneously considered as exemplars, but exchange deterministic messages until a good set of exemplars gradually emerges. AP finds an approximate solution by using this message passing optimization strategy that is based on max-sum algorithm in a factor graph [8]. Hence, AP does not require the number of exemplars, since this number gradually emerges automatically during the process. However AP does require a similarity matrix in main memory as well as a user defined parameter, the preferences, which are the diagonal values of the similarity matrix.

A fast algorithm, which works without the need of any similarity matrix in main memory, is Subtractive Clustering (SC) [9] [10]. This algorithm was also employed in RBF neural network training [11] [12]. Subtractive Clustering can determine both the exemplars and their number [10] but it requires carefully selected user-defined parameters for the bandwidth and the stopping criteria.

In this work we propose a leave-one-out kernel average gradient descent procedure that estimates a bandwidth parameter from the data, and then we use this bandwidth in a modified subtractive clustering algorithm. We demonstrate that the proposed scheme can provide an automatic estimate of most representative exemplars from the data and in the same time can recognize shapes of patterns.

The rest of the paper is organized as follows. Section 2 provides the basics for Subtractive Clustering. Section 3 introduces the proposed gradient descent of the leave-one-out kernel averaged regression function. Section 4 describes all the initializations and the parameter settings for the proposed scheme. Section 5 presents several experimental simulations and comparisons, while section 6 concludes the paper.

2 Subtractive Clustering basics

Subtractive clustering algorithm [9] [10] [11] [12] selects a set of exemplars from the most representative real data points by using their density. Subtractive clustering can work without any priori information about the number of exemplars. In the first step it computes a density-based potential for every point and then gradually subtracts exemplars by updating all the remaining potentials. The potential $P(i)$ for each point $\mathbf{x}_i$ is defined as a sum of Gaussian kernels over all the N data points as:

$$P(i) = \sum_{j=1}^{N} \exp(-a\|\mathbf{x}_j - \mathbf{x}_i\|^2) \quad (1)$$

where $a = (2/\sigma_a)^2$ and the bandwidth σ_a represents a neighbourhood radius. A data point will have high potential $P(i)$ and high density if it has many neighbour points.

After finding all $P(i)$ the algorithm iteratively executes an updating cycle as:

1) Find data point $\mathbf{x}^*$ (cluster center) with the highest potential value P^*

2) Revise the potential of all other points using $P(i) = P(i) - P^*\exp(-b\|\mathbf{x}^* - \mathbf{x}_i\|^2)$

The updating cycle for the potentials $P(i)$ terminates if the current max potential P^* drops below a certain value and the algorithm stops if ($P^* < e\, P_1^*$) [10] [11] [12] where P_1^* is the first max potential and e a small percentage. In each iteration the highest potential P^* of the selected point $\mathbf{x}^*$ will substantially affect all the revised potentials of the points near by. Thus, the data points near the selected point $\mathbf{x}^*$ will have significantly reduced density. The updates of the potentials use $b = (2/\sigma_b)^2$ where bandwidth σ_b is another positive constant which also defines a neighbourhood radius. Usually σ_b is taken to be as $1.5\sigma_a$, in order to avoid the selection of closely located exemplars.

The main problem is choosing an appropriate value for the bandwidth parameter σ_a. This choice is of crucial importance and is usually done via extensive experimentation and trial-and-error. The potentials $P(i)$ represent density. So, one can subjectively try to choose a bandwidth σ_a by looking at potentials produced by a wide range of bandwidths, starting with large values of σ_a and gradually decreasing them until a reasonable density is reached. However, such an approach is impractical and too many validations are needed, since there is no way to define a-priori a suitable density value. This is what we are looking for in the first place. A more important issue is that the potentials affect the number of exemplars and their locations. If the bandwidth is very small this will result in neglecting the effect of neighbouring points and then all points will be selected as exemplars. If the bandwidth is small then many exemplars will be selected. If the bandwidth is large then the density function will be affected by accounting all the points and few exemplars will be selected. If the bandwidth is too large then even fewer exemplars will be selected. It is very easy for anyone to see these limits by using trial-and-error. Furthermore, the bandwidth is dataset dependent and the previous limits depend on the formation of a given dataset. An automatic or semi-automatic process is essential as part of a more global analysis in order to avoid many user-defined parameters. In our scheme the proposed leave-one-out gradient descent provides proper bandwidth values for Subtractive Clustering automatically.

3 Proposed gradient descent of leave-one-out Kernel averaged

We propose gradient descent learning of the kernel averaged (or weighted average) regression function to automatically estimate a bandwidth parameter. Given a training set $\{\mathbf{x}_i, y_i\}_{i=1}^N$ where $\mathbf{x}_i$ are the points and y_j are the desired labels (which we will define later in eq. 4), the conventional kernel averaged regression function $f(\mathbf{x}_i)$ is:

$$f(\mathbf{x}_i) = \sum_k^N g_k(\mathbf{x}_i)\, y_k \tag{2a}$$

$$g_k(\mathbf{x}_i) = \varphi_k(\mathbf{x}_i) / \Big(\sum_j^N \varphi_j(\mathbf{x}_i) \Big) \tag{2b}$$

where $\varphi_k(\mathbf{x}) = \exp(-\delta_k(\mathbf{x})/\sigma^2)$ are Gaussian kernels and $\delta_k(\mathbf{x}) = \|\mathbf{x}_k - \mathbf{x}\|^2$ is the squared Euclidean distance. The kernel averaged $f(\mathbf{x}_i)$ has a nominator $\Sigma\varphi_j(\mathbf{x}_i)y_j$, and a denominator $\Sigma\varphi_j(\mathbf{x}_i)$ defined as a sum of $\varphi_k(\mathbf{x}_i)$ Gaussian kernels over all N data points. Since in subtractive clustering the potential $P(i) = \Sigma\varphi_j(\mathbf{x}_i)$ we can see that actually this potential is the normalization factor of $f(\mathbf{x}_i)$.

3.1 Gradient of the leave-one-out kernel averaged

The proposed *leave-one-out kernel averaged* regression function $f_{loo}(\mathbf{x}_i,\gamma)$ is given by leaving out from the sum in eq. 2 a percentage γ of the self-contribution of $\mathbf{x}_i$ as:

$$f_{loo}(\mathbf{x}_i,\gamma) = \sum_k^N g_k(\mathbf{x}_i)\, y_k - \gamma\, g_i(\mathbf{x}_i)\, y_i \tag{3}$$

where γ is the small leave-one-out parameter which takes values in the range [0, 1].

The proposed method uses desired labels y_i for the points $\mathbf{x}_i$. We define them as:

$$y_i = (1/N) \sum_{j=1}^N \left\| \mathbf{x}_j - \mathbf{x}_i \right\|^2 \tag{4}$$

Thus, each desired label y_i is considered as the variance of the corresponding $\mathbf{x}_i$, if this $\mathbf{x}_i$ was the center of the training set. So

The gradient $\partial E(\sigma,\mathbf{x})/\partial\sigma$, with respect to bandwidth σ, is computed from the squared error $E(\sigma,\mathbf{x})$ which is a convex function defined as $E(\sigma,\mathbf{x}) = (f_{loo}(\mathbf{x},\gamma)-y)^2$ where $f_{loo}(\mathbf{x},\gamma)$ is the leave-one-out kernel averaged regression function.

Without the leave-one-out such a gradient will not work. Taking a gradient of the kernel averaged with respect to the bandwidth will not result in a suitable solution, since eventually all points will converge to tiny bandwidth values (they will be correct for predicting themselves).

The classical squared error $E^i(\sigma,\mathbf{x}_i)$ for each $\mathbf{x}_i$ is:

$$E^i(\sigma,\mathbf{x}_i) = (1/2)\, (f_{loo}(\mathbf{x}_i,\gamma) - y_i)^2 \tag{5}$$

The gradient descent update for the σ parameter can be defined from the gradient of the squared error as:

$$\Delta\sigma = -\,\xi\, \partial E^i(\sigma,\mathbf{x}_i)/\partial\sigma \tag{6}$$

The chain rule of the gradient gives:

$$\partial E^i(\sigma,\mathbf{x}_i)/\partial\sigma = (\partial E^i(\sigma,\mathbf{x}_i)/\partial f_{loo}(\mathbf{x}_i,\gamma))\, (\partial f_{loo}(\mathbf{x}_i,\gamma)/\partial\sigma) = (f_{loo}(\mathbf{x}_i,\gamma) - y_i)\, (\partial f_{loo}(\mathbf{x}_i,\gamma)\, /\partial\sigma) \tag{7}$$

where the derivate $(\partial f_{loo}(\mathbf{x}_i,\gamma)\, /\partial\sigma)$ is:

$$\frac{\partial}{\partial\sigma} f_{loo}(\mathbf{x}_i,\gamma) = \sum_k^N \left(\frac{\partial}{\partial\sigma} g_k(\mathbf{x}_i)\, y_k \right) - \gamma \frac{\partial}{\partial\sigma} g_i(\mathbf{x}_i)\, y_i \tag{8}$$

where we only need to find the derivate $\partial g_k(\mathbf{x}_i)/\partial\sigma$ given by:

$$\begin{aligned} \frac{\partial}{\partial\sigma} g_k(\mathbf{x}_i) &= \frac{\partial}{\partial\sigma} \left[\varphi_k(\mathbf{x}_i) \cdot \Big(\sum_j^N \varphi_j(\mathbf{x}_i) \Big)^{-1} \right] = \\ &= \Big(\frac{\partial}{\partial\sigma} \varphi_k(\mathbf{x}_i) \Big) \cdot \Big(\sum_j^N \varphi_j(\mathbf{x}_i) \Big)^{-1} - \varphi_k(\mathbf{x}_i) \cdot \Big(\sum_j^N \varphi_j(\mathbf{x}_i) \Big)^{-2} \Big(\sum_j^N \frac{\partial}{\partial\sigma} \varphi_j(\mathbf{x}_i) \Big) \end{aligned} \tag{9}$$

This equation by using $\frac{\partial}{\partial\sigma}\varphi_k(\mathbf{x}_i) = \varphi_k(\mathbf{x}_i)\,\delta_k(\mathbf{x}_i)\,/\sigma^3$ becomes:

$$\frac{\partial}{\partial\sigma} g_k(\mathbf{x}_i) = \left(\varphi_k(\mathbf{x}_i)\delta_k(\mathbf{x}_i)/\sigma^3\right)\cdot\left(\sum_j^N \varphi_j(\mathbf{x}_i)\right)^{-1} - \varphi_k(\mathbf{x}_i)\cdot\left(\sum_j^N \varphi_j(\mathbf{x}_i)\right)^{-2}\left(\sum_j^N \left(\varphi_j(\mathbf{x}_i)\,\delta_j(\mathbf{x}_i)/\sigma^3\right)\right)$$

$$= (1/\sigma^3)\,\varphi_k(\mathbf{x}_i)\left(\sum_j^N \varphi_j(\mathbf{x}_i)\right)^{-1}\left[\delta_k(\mathbf{x}_i) - \left(\sum_j^N \varphi_j(\mathbf{x}_i)\right)^{-1}\left(\sum_j^N \left(\varphi_j(\mathbf{x}_i)\,\delta_j(\mathbf{x}_i)\right)\right)\right] \quad (10)$$

and by replacing the expression for $g_k(\mathbf{x}_i)$ from eq.2b into eq. 10 it gives:

$$\frac{\partial}{\partial\sigma} g_k(\mathbf{x}_i) = (1/\sigma^3)\,g_k(\mathbf{x}_i)\left[\delta_k(\mathbf{x}_i) - \left(\sum_j^N \left(g_j(\mathbf{x}_i)\,\delta_j(\mathbf{x}_i)\right)\right)\right] \quad (11)$$

Eq. 11 is the general derivate for any function $g_k(\mathbf{x}_i)$.

The derivate $\partial g_i(\mathbf{x}_i)/\partial\sigma$ (of the contribution of $\mathbf{x}_i$ to itself) has a shorter expression produced by eq. 11 which after simplifications (by setting $\delta_i(\mathbf{x}_i) = 0$ and $\varphi_i(\mathbf{x}_i) = 1$) is:

$$\frac{\partial}{\partial\sigma} g_i(\mathbf{x}_i) = -(1/\sigma^3)\left(\sum_j^N \left(\varphi_j(\mathbf{x}_i)\,\delta_j(\mathbf{x}_i)\right)\right)\left(\sum_j^N \varphi_j(\mathbf{x}_i)\right)^{-2} \quad (12)$$

Finally by substituting eq. 11 and eq. 12 into eq. 8 we can compute $(\partial f_{loo}(\mathbf{x}_i,\gamma)/\partial\sigma$. In a more shorthanded notation it gives:

$$\frac{\partial}{\partial\sigma} f_{loo}(\mathbf{x}_i) = (1/\sigma^3)\sum_k^N \left(\left(\delta_k - \frac{\Sigma(\varphi_j\delta_j)}{\Sigma\varphi_j}\right)\cdot\frac{\varphi_k}{\Sigma\varphi_j}\,y_k\right) + \gamma\frac{\Sigma(\varphi_j\delta_j)}{(\Sigma\varphi_j)^2}\,y_i \quad (13)$$

The small leave-one-out parameter $\gamma \in [0, 1]$ prevents the gradient from converging into tiny values of the bandwidth σ. There exists a trade-off between γ=1 which gives large bandwidths and γ=0 which gives tiny bandwidths.

Stochastic mode (or online) of gradient descent learning computes the gradient by using a single example at a time. The algorithm randomly selects an example $\mathbf{x}_i$ and its label y_i and updates the current parameter σ by using:

$$\sigma^{(t+1)} = \sigma^{(t)} - \xi\,\partial E(\sigma^{(t)}, \mathbf{x}_t)/\partial\sigma \qquad t = 1,\ldots N \quad (14)$$

Hence, an epoch ends after all examples are introduced in a random order. Then the gradient updates of σ are averaged over all N examples as $\sigma^{epoch} = \text{avg}(\sigma^{(t)})$ with t=1,…,N. The learning rate ξ can be constant or can vary at each epoch. For one epoch step the *leave-one-out kernel averaged gradient descent* is:

for $t = 1$ **to** N
  pick randomly a point $\mathbf{x}_t$ without replacement
  update the parameter σ by using $\sigma^{(t+1)} = \sigma^{(t)} - \xi\,\partial E(\sigma^{(t)}, \mathbf{x}_t)/\partial\sigma$
end for

4 Initializations and parameter settings

As usual the first thing to do is to scale the data features into the range [0, 1]. Without scaling the gradient might not converge, since the learning rate ξ depends on the scale of the feature space. By scaling the data features first, we can then use a fixed value for ξ for all datasets and hence avoid searching for suitable learning rates each time we use a different dataset. Such scaling also avoids over-fitting which occurs when some features are in large numeric ranges.

In Subtractive Clustering (SC) the potential updating cycle terminates if the current max potential P^* become less that a threshold ($P^* < e\ P_1^*$). If e is selected to be very small, a large number of exemplars will be selected. On the contrary, a large value of e will lead to a small exemplar set. In order to avoid any other user-defined parameter we set $e = 1/P_1^*$. That is, Subtractive Clustering terminates at j-th iteration when $P_j^* < 1$. Thus, every point starts with potential $P(i) >= 1$ and finally ends up with potential $P(i) < 1$. There is a theoretical justification for this limit since $P(i)=1$ is the self-contribution of every i-th point to itself.

For 2-dimensional datasets in Subtractive Clustering we set $\sigma_b = 1.5\sigma_a$ as recommended. High dimensional density estimates may suffer from the curse of dimensionality. For higher dimensions there is a problem since the 1.5 percentage influences more strongly the nearby points and we use a variable $\sigma_b = \sigma_a + 0.5\ (1.0 - k_{\text{sofar}}/N)\ \sigma_a$, which starts from $\sigma_b = 1.5\sigma_a$ and decays. As k_{sofar} (the number of selected exemplars so far) increases from 1 to k during the $P(i)$ updating cycle of SC, the parameter σ_b gradually decreases and in the theoretical limit $k=N$ the value σ_b becomes equal to σ_a.

For the online gradient descent we set a fixed learning rate $\xi = 0.2$ and maximum epochs = 10. Usually it converges after the first epoch if the dataset size is larger than 10000. So, for larger datasets we can set maximum epochs = 2.

For the leave-one-out kernel averaged regression function we set the leave-one-out parameter $\gamma = 0.1$. The value $\gamma = 1$ removes the self-contribution completely and will give a large bandwidth and very few exemplars, while $\gamma = 0$ will give a tiny bandwidth and almost all points as exemplars. Since the goal is just to avoid this, we found after some experimentation that a value $\gamma = 0.1$ is always sufficient enough to prevent bandwidth from converging into tiny values, so as to provide a stable solution without producing large bandwidths.

Initializing the bandwidth σ in the beginning of gradient descent (epoch = 0) is an issue, since for different datasets we may need to search for different initial values of σ each time. However there is a simple automatic way that works around this. We set the initial bandwidth equal to the trace of covariance matrix $\mathbf{R}$. Hence, given N points $\mathbf{x}_n$ each one in d dimension, with their mean $\boldsymbol{\mu} = (1/N) \sum_n^N \mathbf{x}_n$ the covariance matrix is $\mathbf{R} = (1/N) \sum_n^N (\boldsymbol{\mu} - \mathbf{x}_n)\ (\boldsymbol{\mu} - \mathbf{x}_n)^{\mathrm{T}}$ and the initial value of σ is $(1/d) \sum_i^d \sqrt{r_{ii}}$, where r_{ii} are the diagonal elements of $\mathbf{R}$. Thus, the gradient descent starts with a relative large bandwidth σ which decreases immediately after the first epoch, until it converges.

It is important to note that we use the same settings for all the datasets and no user-defined parameter is needed.

5 Experimental simulations

The first set of experimental simulations present results for visual comparisons of AP with the proposed algorithm using four 2-d datasets. The second set present performance comparisons and quality analysis on several real world benchmark datasets.

The code for Affinity Propagation (AP) was downloaded from the official site (http://www.psi.toronto.edu/affinitypropagation). AP uses as input a similarity matrix S in which the pair-wise similarities between data points are defined from their distances as $s(i,k) = -\|\mathbf{x}_i - \mathbf{x}_k\|^2$ for every $i \neq k$, as suggested in [8]. There are two more parameters: the damping factor λ and the prior preferences $s(k,k)$ which are the diagonal values of the similarity matrix. The dumping factor is usually $\lambda = 0.5$ as suggested. For the preferences, a good choice [8] is to set all the diagonal elements $s(k,k)$ equal to the median value of all the similarities between data points. We use as preference the one half of the mean value of all similarities $(1/(2N^2)) \sum_i^N \sum_k^N s(i,k)$ that results in a moderate number of exemplars which emerge automatically. This choice selects much more exemplars than the median choice while it still avoids selecting outliers.

5.1 Evaluation Criteria and Quality indexes

The *sum of squared errors* (*SSE*) which quantifies the *clustering error* is the most widely used quality criterion [8] and is given by the sum of the squared distance between each point $\mathbf{x}_i$ and its corresponding exemplar $\mathbf{c}(\mathbf{x}_i)$ as:

$$SSE = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \mathbf{c}(\mathbf{x}_i) \right\|^2 \tag{15}$$

The *maximum distance* (*maxD*) between any point $\mathbf{x}_i$ and its exemplar $\mathbf{c}(\mathbf{x}_i)$ that can quantify if all points are compactly represented (no cluster is larger than *maxD*) is:

$$maxD = \max_i^N \|\mathbf{x}_i - \mathbf{c}(\mathbf{x}_i)\|^2 \tag{16}$$

The *normalized Hubert gamma statistic* [13] is a well known cluster evaluation criterion which is invariant to the number of clusters, given by:

$$\hat{\Gamma} = \frac{\frac{1}{M} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (P(i,j) - \mu_P) \times (Q(i,j) - \mu_Q)}{\sigma_P \times \sigma_Q} \tag{17}$$

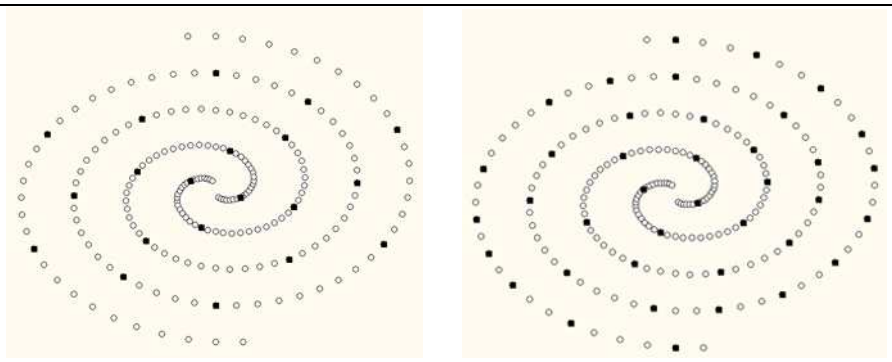
where $M = N(N-1)/2$, and it uses two proximity matrices P and Q both of size $N \times N$. An element $P(i, j)$ is the distance between points $\mathbf{x}_i$ and $\mathbf{x}_j$. An element $Q(i, j)$ is the distance between the cluster representative centroids to which $\mathbf{x}_i$ and $\mathbf{x}_j$ belong. μ_P is the mean of all elements of matrix P, μ_Q is the mean of all elements of matrix Q, while σ_P and σ_Q are their standard deviations from their means. A high value of this statistic (close to 1) indicates the existence of well-separated compact clusters.

The *net similarity cost* is defined as a cost function specifically for AP [8] [14] and it is the sum of similarities $s(i,k)$ between data points and their exemplars, minus the exemplar costs $s(k,k)$, (the preferences of the exemplars). AP identifies a set of exemplars K so as to maximize this cost given by [14]:

$$\sum_{i \notin K} \max_{k \in K} s(i,k) + \sum_{k \in K} s(k,k) \tag{18}$$

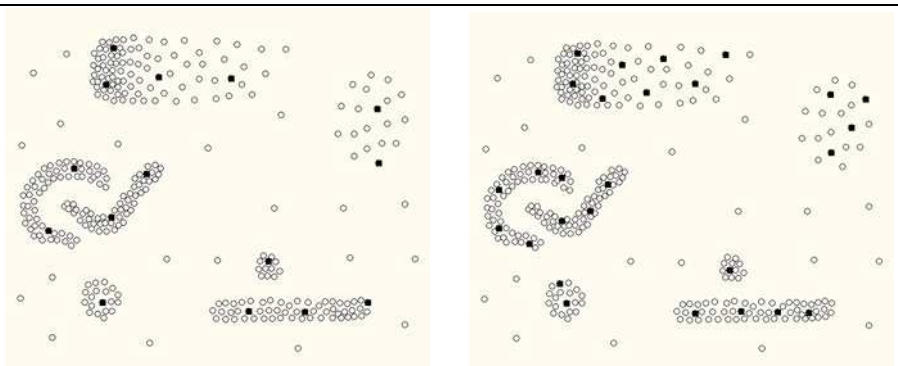
5.2 Visual comparisons of AP with the proposed KG-SC

For the visual comparisons we use four datasets with 2 dimensions each. We compare the results of Affinity Propagation (AP) algorithm with the proposed leave-one-out kernel gradient subtractive clustering (KG-SC in short).



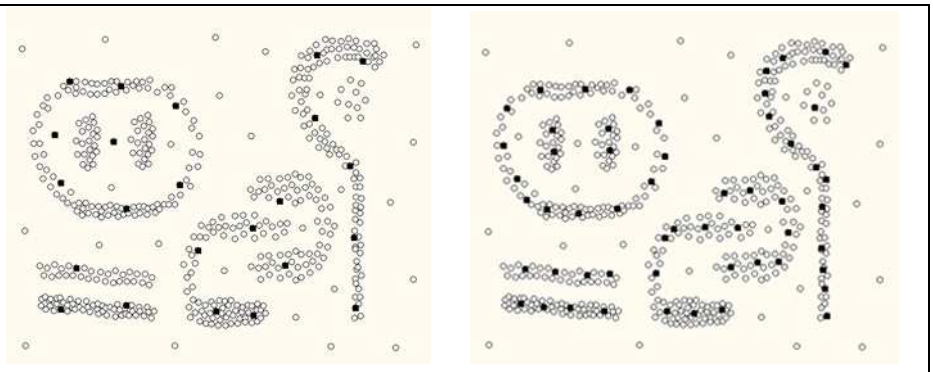
AP found 20 exemplars automatically KG-SC found 39 exemplars automatically

Fig. 1. Dataset 1 (two spirals) has 200 points. Exemplars are marked as black squares, while the other points are marked as white circles.



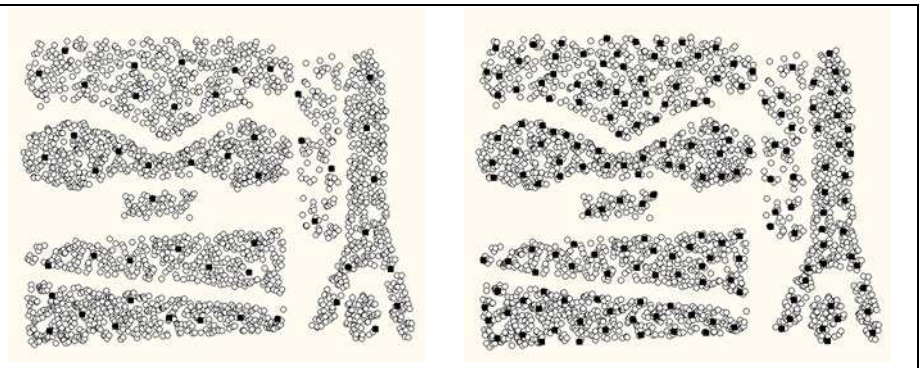
AP found 15 exemplars automatically KG-SC found 27 exemplars automatically

Fig. 2. Dataset 2 has 322 points. Exemplars are black squares, other points are white circles.



AP found 23 exemplars automatically KG-SC found 55 exemplars automatically

Fig. 3. Dataset 3 has 523 points. Exemplars are black squares, other points are white circles.



AP found 50 exemplars automatically KG-SC found 179 exemplars automatically

Fig. 4. Dataset 4 has 2551 points. Exemplars are black squares, other points are white circles.

Table 1 illustrates the quality indexes that correspond to the exemplar selections and clustering solutions of the datasets in figs 1-4 for the AP and KG-SC.

Table 1. Quality indexes for the exemplar selection and clustering solutions of the datasets in figs 1-4. For each algorithm (AP and KG-SC) we illustrate the net similarity cost, maximum Distance, clustering error, normalized Hubert statistic. Best indexes are marked in bold.

		Affinity propagation algorithm				Proposed leave-one-out kernel gradient subtractive clustering			
	N	*netSim*	*maxD*	*error*	*Hubert*	*netSim*	*maxD*	*error*	*Hubert*
Dataset 1	200	**−3.55**	0.028	1.49	0.915	−4.63	**0.016**	**0.62**	**0.979**
Dataset 2	322	**−4.01**	0.061	1.61	0.959	−5.49	**0.058**	**1.03**	**0.974**
Dataset 3	523	**−5.84**	0.058	2.41	0.954	−9.23	**0.042**	**0.89**	**0.980**
Dataset 4	2551	**−14.4**	0.015	6.61	0.976	−31.2	**0.003**	**1.44**	**0.995**

From table 1 it seems that both algorithms can provide high quality results for the 2-dimensional datasets, while a slight precedence could be given to KG-SC.

In addition, it is apparent in table 1 that AP delivers exactly what it promises, that is to identify a set of exemplars *K* so as to maximize the net similarity cost [14]. The net similarity cost is better for AP than KG-SC. So AP remains the best algorithm for the *k*-centers problem.

However the clustering error that quantifies the distortion and the normalized Hubert index that quantifies the cluster compactness are better for KG-SC. Note that ideal clustering solutions usually have the normalized Hubert index close to 1 as they are in the last column of table 1. So KG-SC delivers more well defined exemplars.

It is the *k*-centers problem itself which might not be able to guarantee the best exemplar selection. That is why KG-SC takes a different path; the density based, and tries to find the most important representatives from the densest ones. The better quality of the KG-SC solutions is evident from the maximum Distance, clustering error and normalized Hubert statistic in table 1.

The computational complexity cost of the proposed KG-SC is quadratic $O(N^2)$ of the same order with the cost of SC. Actually, for large datasets and max epochs = 2 it is two times that of SC. This cost is much lower than the AP cost. The memory requirements for KG-SC is $O(Nd)$, since only the dataset is needed in main memory. On the other hand, AP does require three matrices (similarities, availabilities, responsibilities) of size $N \times N$ in main memory and this could limit the algorithm. However, one can argue that for the special case of ultra high-dimensional datasets where the data dimension is of the same order with the number of examples ($d \approx N$) the memory requirements become the same.

What will happen in a case where someone needs a fixed number of exemplars less than the KG-SC algorithm finally selects is a question that could be answered. Note that an advantage of Subtractive Clustering is that it returns exemplars in decreasing order from the most important to the least important. So, picking the first *K* in this list is one simple solution.

5.3 Quality Comparisons on real world benchmark datasets

Quality comparisons are also performed on a number of publicly available real-world benchmark problems which are downloaded from the UCI machine learning data repository (http://archive.ics.uci.edu/ml). The specific details of these datasets (dataset name, *N* examples before duplicate removal, *d* dimensions) are illustrated in table 2 together with the results.

We found that while KG-SC as a density-based algorithm does not suffer from the existence of duplicates, AP does. Thus for a fair comparison we first remove all duplicates from the benchmark datasets. Also, since the net similarity is not a quality index but a specific cost suitable only for AP (it was always better for AP) we do not illustrate it in table 2. Note, for future considerations that we detect several duplicates in the datasets Banknote Authentication, Blood Transfusion, Phoneme, Wisconsin Breast Cancer, Haberman, Yacht Hydrodynamics, Red Wine Quality, White Wine Quality, Concrete Compressive Strength.

Table 2. Quality indexes for the exemplar selection of various benchmark datasets with *N* examples and *d* dimensions. For each algorithm (AP and KG-SC) we illustrate the *k* number of selected exemplars which emerge automatically, the maximum Distance, the clustering error and the normalized Hubert gamma statistic. Best quality indexes are marked in bold.

			Affinity Propagation algorithm				Proposed leave-one-out kernel gradient subtractive clustering			
name	*N*	*d*	*k*	*maxD*	*error*	*Hubert*	*k*	*maxD*	*error*	*Hubert*
CPU	209	7	*21*	0.93	4.69	0.953	*20*	**0.60**	**4.33**	**0.960**
Yacht	308	6	*33*	0.14	7.73	0.958	*54*	**0.05**	**3.37**	**0.980**
Housing	506	13	*35*	0.66	39.9	0.965	*45*	**0.61**	**37.8**	**0.967**
Concrete	996	8	*80*	0.38	30.4	0.943	*114*	**0.18**	**25.0**	**0.960**
Airfoil	1503	6	*79*	0.19	20.7	0.980	*138*	**0.05**	**11.2**	**0.989**
Abalone	4177	7	*127*	0.12	**16.2**	**0.991**	*45*	**0.11**	21.9	**0.991**
RedWine	1599	11	*116*	0.65	**50.0**	**0.876**	*104*	**0.53**	56.2	0.857
Bodyfat	252	15	*29*	**0.28**	**18.7**	**0.923**	*21*	1.59	24.8	0.759
WhiteWine	4898	11	*292*	**0.11**	**77.6**	**0.917**	*265*	0.47	92.4	0.886
Iris	150	4	*10*	0.14	2.95	0.962	*22*	**0.09**	**1.61**	**0.978**
Haberman	306	3	*23*	0.20	4.02	0.907	*47*	**0.17**	**2.37**	**0.938**
Ecoli	336	7	*27*	0.21	9.29	0.931	*44*	**0.20**	**7.45**	**0.949**
Blood Trans	742	4	*28*	0.35	4.13	0.948	*34*	**0.11**	**3.28**	**0.963**
Banknote	1372	4	*55*	0.06	7.38	0.977	*115*	**0.05**	**3.58**	**0.989**
Phoneme	5404	5	*165*	0.12	32.3	0.950	*374*	**0.07**	**20.5**	**0.969**
Yeast	1484	8	*107*	**0.11**	**22.3**	**0.960**	*119*	0.17	23.2	**0.960**
Diabetes	768	8	*83*	**0.18**	**30.6**	**0.895**	*65*	0.62	39.1	0.834
Wine	178	13	*27*	**0.54**	**26.6**	**0.885**	*18*	0.98	37.6	0.828
Heart	297	13	*43*	**1.38**	**92.1**	**0.904**	*25*	2.13	138	0.855
Wisconsin	683	9	*37*	**0.97**	**83.9**	**0.931**	*27*	1.93	109	0.909
Dermatology	358	34	*41*	**2.50**	**320**	**0.909**	*36**	4.34	371	0.877
Shuttle	58000	9	*	*****	*****	*****	*956*	**0.002**	**1.01**	**0.999**

Both algorithms find well defined representative exemplars and deliver high quality solutions, since the normalized Hubert gamma index is very high in both of them for all benchmark datasets. In low dimensional datasets KG-SC seems better, while in high dimensional datasets AP seems better.

There are some limitations. AP has limits in the number of examples, while the proposed KG-SC is density-based and might limited by the number of dimensions (features). The Dermatology dataset has many dimensions ($d = 34$) and the leave-one-out gradient could not converge for $\gamma = 0.1$, so we use a minimum value $\gamma = 0.01$. The Shuttle dataset has quite many examples ($N = 58000$) and the Affinity Propagation runs out of memory (it needs 39 GB). For the Shuttle dataset the proposed leave-one-out Kernel Gradient Subtractive Clustering produces 956 exemplars and a normalized Hubert gamma statistic 0.999 which indicates very well formed compact clusters.

6 Conclusions

We present a scheme that can potentially permit automatic selection of representative exemplar points from the data without the need of any used-defined parameter. By computing a gradient descent for a simple leave-one-out kernel averaged regression function that can automatically estimate a suitable bandwidth parameter for the density-based Subtractive Clustering algorithm we can extract most representative exemplars, without initial knowledge of their number. Evaluating with classical quality indexes the data clustering solutions around these exemplars reveal that the proposed KG-SC algorithm produce well separated compact and dense clusters. Experimental comparisons with the state-of-the-art Affinity Propagation exemplar selection algorithm show that both algorithms select well defined representative exemplars and can deliver high quality solutions. KG-SC is simply parallelizable, a point worthwhile studying in the future. We also plan to explore the possibility of using either mini-batch gradients, or a dual tree for speeding up KG-SC. Interesting future works could extend KG-SC in order to explore a possible automation in other density based algorithms. Currently we study the proposed KG-SC for training Neural Networks.

References

1. Dunham, M.H.: Data mining introductory and advanced topics. Prentice Hall (2004).
2. Duda, R., Hart, P.E., Stork D.G.: Pattern Classification, John Wiley & Sons (2001)
3. Mézard, M.: Where are the exemplars? Science, 315, 949–951 (2007)
4. Kaufman, L., Rousseeuw, P.: Clustering by means of medoids. In Statistical Data Analysis Based on the L1 Norm and Related Methods, 405–416 (1987)
5. Kariv, O., Hakimi, S.L: An algorithmic approach to network location problems. ii: The p-medians. Siam Journal on Applied Mathematics, 37, (1979)
6. Hochbaum, D., Shmoys, D.: A best possible heuristic for the k-center problem. Mathematics of Operations Research, 10(2), 180–184, (1985)
7. Bern, M., Eppstein, D.: Approximation algorithms for NP-hard problems. PWS, Publishing, Chapter Approximation algorithms for geometric problems, 296–345 (1997)
8. Frey B.J., Dueck, D.: Clustering by passing messages between data points. Science, 315, 972–976 (2007)
9. Chiu, S.L.: Fuzzy Model Identification Based on Cluster Estimation. Journal of Intelligent and Fuzzy Systems 2, 267–278 (1994).
10. Kothari, R., Pittas, D.: On finding the number of clusters. Pattern Recognition Letters (1999)
11. Sarimveis, H., Alexandridis, A. Bafas, G.: A fast training algorithm for RBF networks based on subtractive clustering. Neurocomputing 51, 501–505 (2003)
12. Yang, P., Zhu, Q., Zhong, X.: Subtractive Clustering Based RBF Neural Network Model for Outlier Detection. Journal of Computers, 4(8), 755–762 (2009)
13. Halkidi, M., Batistakis, Y., and Vazirgiannis, M. On Clustering Validation Techniques. Journal of Intelligent Information Systems, 17:2/3, 107–145 (2001)
14. Dueck, D., Frey, B., Jojic, N., Jojic, V., Giaever, G., Emili, A., Musso, G., Hegele, R.: Constructing treatment portfolios using affinity propagation. In: vol. 4955 of Lecture Notes in Computer Science, pp. 360–371, Springer Berlin/Heidelberg (2008)